现代JS: https://zh.javascript.info/

Object常用方法:

- Object.assign() 方法用于将所有可枚举属性的值从一个或多个源对象分配到目标对象。它将返回目标对象
- Object.entries()方法返回一个给定对象自身可枚举属性的键值对数组，其排列与使用 for...in 循环遍历该对象时返回的顺序一致
- hasOwnProperty() 方法会返回一个布尔值，指示对象自身属性中是否具有指定的属性
- Object.is() 方法判断两个值是否为同一个值
- Object.freeze() 方法可以冻结一个对象。一个被冻结的对象再也不能被修改；冻结了一个对象则不能向这个对象添加新的属性，不能删除已有属性，不能修改该对象已有属性的可枚举性、可配置性、可写性，以及不能修改已有属性的值
- Object.seal()方法封闭一个对象，阻止添加新属性并将所有现有属性标记为不可配置
- Object.create()方法创建一个新对象，使用现有的对象来提供新创建的对象的

```
var obj = new Object();
   console.log(obj);
//对象的合并
var obj1 = {name:'ABC', age:19};
var obj2 = {province:'贵州', city:'六盘水'};
var result = Object.assign(obj1, obj2);
    console.log(obj1);
    console.log(result);
    console.log(obj1 === result);

//生成可遍历的数组
var objArr = Object.entries(obj1);
    console.log(objArr);
    console.log(Object.values(obj1));
    console.log(Object.keys(obj1));

 //检查对象是否包含指定的属性
    console.log(obj1.hasOwnProperty('name'));//true
    console.log(obj1.hasOwnProperty('valueOf'));//false  不包括继承过来的

//is()
    console.log(Object.is(NaN, NaN));//true
    console.log(Object.is(0, -0));//false
```

```
    console.log(Object.is(0, -0.0));//false
    console.log(Object.is(0, 0.0));//

//冻结一个对象  ==>定义一个静态的对象
var obj = {a:1, b:2};
    Object.freeze(obj);
    delete obj.a; //删除不掉
    obj.c = 3;//加不上
    obj.a = 100; //改不了
    console.log(obj);
    console.log(Object.isFrozen(obj));//true

//封闭一个对象
var obj2 = {a:100, b:200};
    Object.seal(obj2);
    obj2.a = 600;//对可以修改的属性是可以进行修改的
    obj2.c = 800;//但是不能增减属性
    delete obj2.b;
    obj2.b = 5000;
    console.log(obj2);

```

Object.prototype.valueOf()

Object.prototype.valueOf()：返回当前对象对应的值，如下表所示。valueOf()可通过实例直接调用

对象	返回值
Array	返回数组对象本身。
Boolean	布尔值。
Date	存储的时间是从 1970 年 1 月 1 日午夜开始计的毫秒数 UTC。
Function	函数本身。
Number	数字值。
Object	对象本身。这是默认情况。
String	字符串值。
	Math 和 Error 对象没有 valueOf 方法。

Object.prototype.toString()

Object.prototype.toString()是返回一个对象的字符串形式

Object的toSting()方法并没有什么实际用途，但是实例自定义的方法，如字符串、数组等自定义的toSting()方法能够得到实例转换成字符串的形式，返回的都是字符串

```
console.log(new Object().toString());  //[object Object]
console.log([1, 2, 3, 'aa'].toString());  //1,2,3,aa
console.log('ssss'.toString());   //ssss
console.log(true.toString());  //true
console.log(new Date().toString()); // Fri Apr 22 2022 13:29:38 GMT+0800 (中国标准时间)
```

Object.prototype.toString.call()

console.log(new Object().toString()); //[object Object]中返回的后一个Object是指定的该值的构造函数，因此根据这个方法可以用于判断数据类型的方法。但是实例对象自定义的toString()方法会覆盖Object.prototype.toString方法，因此需要使用call来改变this，通过Object.prototype.toString.call()方法可以判断值的类型。

```
console.log(new Object().toString());  //[object Object]
console.log(Object.prototype.toString.call([1, 2, 3, 'aa']));  //[object Array]
console.log(Object.prototype.toString.call('ssss'));   //[object String]
console.log(Object.prototype.toString.call(true));  //[object Boolean]
console.log(Object.prototype.toString.call(new Date()));  //[object Date]
```

Object.prototype.hasOwnProperty()

Object.prototype.hasOwnProperty()：该方法会返回一个布尔值，指定对象自身属性中是否存在指定的属性，**并且这个属性是非原型链上继承的**，in运算符会获取从原型链上继承的属性。只要有该属性便返回true，即使该属性值为null或undefined

```
let person = {
 name: 'simon',
 age: 24,
    job: undefined,
    girlFriend: null
};
console.log(person.hasOwnProperty('name'));  //true
console.log(person.hasOwnProperty('job')); //true
```

```
console.log(person.hasOwnProperty('girlFriend')); //true
console.log(person.hasOwnProperty('toString'));  //false
```

Object.prototype.isPrototypeOf()

prototypeobj.isPrototypeOf(object)：用于判断object对象是否存在于另外prototypeobj对象的原型链上，返回一个布尔值。

```
function Foo() {}
function Bar() {}
function Baz() {}

Bar.prototype = Object.create(Foo.prototype);
Baz.prototype = Object.create(Bar.prototype);

var baz = new Baz();

console.log(Baz.prototype.isPrototypeOf(baz)); // true
console.log(Bar.prototype.isPrototypeOf(baz)); // true
console.log(Foo.prototype.isPrototypeOf(baz)); // true
console.log(Object.prototype.isPrototypeOf(baz)); // true
```

instanceof运算符

instanceof运算符 用于检测构造函数的 **prototype** 属性是否出现在某个实例对象的原型链上，返回一个布尔值。

```
// 定义构造函数
function C(){}
function D(){}

var o = new C();

o instanceof C; // true，因为 Object.getPrototypeOf(o) === C.prototype

o instanceof D; // false，因为 D.prototype 不在 o 的原型链上

o instanceof Object; // true，因为 Object.prototype.isPrototypeOf(o) 返回 true
C.prototype instanceof Object // true，同上
```

```
13
14 C.prototype = {};
15 var o2 = new C();
16
17 o2 instanceof C; // true
18
19 o instanceof C; // false，C.prototype 指向了一个空对象,这个空对象不在 o 的原型链上.
20
21 D.prototype = new C(); // 继承
22 var o3 = new D();
23 o3 instanceof D; // true
24 o3 instanceof C; // true 因为 C.prototype 现在在 o3 的原型链上
```